

JavaScript for React

React Workshop - P1

Brief History of JavaScript

- Created at Netscape in 1995
- Standardized by ECMA from 1997
 - ES₁ - 1997
 - ES₂ – 1998
 - ES₃ – 1999
 - ES₄ – Never came out
 - ES₅ – 2009
 - ES₆ – 2015
 - ES₇ – 2016
 - ESNext – Language proposals under discussion
- Introduction of NodeJS made JavaScript real software language

Compatibility & Transcompilation

- Not all features of ES7 (or even ES6) are available across all browsers or NodeJS
- Compatibility Guide - <http://kangax.github.io/compat-table/es2016plus/>
- Babel

Declaring Variables

- The 'const' keyword
 - A constant is a variable that can't be overwritten
 - It will generate an error, if we try to overwrite the value
- The 'let' keyword
 - Provides block scope to variable
 - The 'var' keyword provided function scope
- Variables declared with 'const' and 'let' are not hoisted

Template Strings

- Also called 'Template Literals' or 'String Templates'
- Template Strings respects whitespaces
- Suitable for defining mail templates or for embedding HTML in JavaScript code

Arrow Functions

- Non verbose way of defining functions
- Suitable for defining callback / anonymous functions
- Provides lexical scope for 'this' object

Objects & Arrays

- ES7 provided few creative ways of scoping variables within Objects & Arrays
- De-structuring Objects
- De-structuring Arrays
 - List Matching
- Object Literal Enhancement
 - We can grab variables from global scope and add it into object
- Spread Operator

The Fetch API

- Fetch allows you to make network requests similar to XMLHttpRequest (XHR)
- Fetch API uses Promises API, which enables simpler cleaner API, avoiding callback hell
- With Fetch, we achieves what jQuery.ajax provides at the language level. (And much more)

Asynchronous JavaScript

- Promise
 - *"Promise is a promise of a value in future!"*
 - The promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value
- Async/Await
 - Popular way of handling Promises
 - Async/Await code looks deceptively imperative

Classes

- In ES6, classes were introduced to facilitate class based Object Oriented Programming
- ES6 classes are just syntactic sugar over prototype based object modelling
- JavaScript still is a Prototype based Object Oriented Language

Modules

- Introduced in ES6
- Module is a piece of reusable code that can easily be incorporated into other JavaScript files
- Modules are stored in separate files, one file per module
- 'export'/'import' keywords facilitate modules, but these keywords are not fully supported in all browsers or in Node

Assignments

1. Compilation of workshop examples.
2. Write method to do shallow copy of a list using ES6 feature.
3. Write method to do shallow copy of an object using ES6 feature.
4. Write method to do deep copy of a list without using any third party library.
5. Write method to do deep copy of a object without using any third party library.
6. Is the Fetch API an ECMAScript feature ?

Appendix 1 – Scope of this in Arrow Functions

- Before arrow functions, every new function defined its own this value based on how the function was called - "Dynamic Name Binding"
 - A new object in the case of a constructor
 - undefined in strict mode function calls
 - 'window' object in non-strict mode function calls
 - The base object if the function was called as an 'object method'
- Arrow functions don't have their own this binding. They inherit them from their parent lexical scope - "Lexical Name Binding"